An XVM named Edo 6

Ecensity *

December 16, 2005

Abstract

Network applications producing and consuming XML formats benefit heavily from general XML programming languages. The software filling this rôle is taking the shape of network client and server systems being called XVMs [5].

The Ecensity XVM, internally Edo 6, presents a regular and extensible syntax and semantics for XML programming languages with the added benefits derived from a design program founded in the domain of *interaction computing* [9, 10].

1 XVM

The consumption, transformation and production of XML data formats benefits from an XML programming language [1] that can *in- line* both data and code for the production of output, or the recognition of input. These data- code expressions permit the XVM to consolidate programmers' needs into one domain where application problems are solved once and available everywhere. A single programming language (ie, syntax) for application development translates directly into flexibility for programmers, and potentially for their application users.

Li [5], et al., review the need for a regular and extensible interpretation of XML programming languages as the research statement for their work into an XVM at IBM. Li, et al., focus on the binding from XML to XVM implementation, and XVM application components. The former delivers extensibility, and the latter is of course critical to sensible architectures for complex applications.

The Ecensity Edo 6 takes the XVM one important and substantial step beyond this nascent tradition, building on the power of *interaction computing* [8, 9, 10, 11].

2 Interaction Machine

The famous Tape Machine defined by Alan Turing has a sibling which Turing called the Choice Machine, and Peter Wegner formalizes as the Interaction Machine. First, the interaction machine is an open system where the tape machine is closed. It is open to interaction with human users and other machines during the execution of its programming [7, 4, 3, 2, 6].

In most ways the point is extremely rare and subtle, as for many decades software systems are interactive while computing hardware and programming languages have evolved too little from the foundations of Computer Science laid down by Turing's Tape Machine. The proof of this statement is the number of bugs common to all software systems: reasoning about interactive systems is hard because our programming languages are a poor human- computer interface for non- Tape Machine type applications. The evolution of computing hardware and programming languages needs to make the production of their interactive applications more reasonable for their programmers, for fewer defects (bugs) for their users.

Interaction is more powerful than algorithms [9], for the reason that Turing's Tape and Choice machines are not logically equivalent. The evaluation of the interactive program is *informed* by its interactions. Employing these asynchronous events, most explicitly in distributed systems (automata), is a network centric perspective for both the designer and

* *http://www.ecensity.com/*

programmer of interactive languages.

3 Edo 6

The Ecensity XVM has been designed for regular semantics over general classes of programming language operators (rop, control and fail type operators), for machine generated code, broad extensibility, and to exploit the simple tree structure of XML syntax for the reasonable analysis of its programs – eg, application security analysis. Rationalizing these objectives produced a machine language, which has the benefit of being semantically simple and easy to learn while using a debugger that visualizes the state of the XVM.

The Edo 6 programming language is very abstract, making it very productive. Its level of abstraction is appropriate to an interaction language with first class network references. For example, a complete network transaction is implied by the `node:push` operation in the following example.

```
<edo:init xmlns:edo="ns:com.exocen.edo"
          xmlns:core="ns:com.exocen.core"
          xmlns:node="ns:com.exocen.Node">

 <node:push src="http://host/service.xml"/>
 <core:echo/>
</edo:init>
```

This example pushes the XML document node set requested from a service onto the operand- return stack, and then outputs the reformatted XML document to the user output. The `node:push` onto the operand- return (ROP) stack is consumed by `core:echo`.

The `edo:init` plain vanilla programming model is complemented by output production `edo:page`, and input recognition operators `edo:model` and `edo:style`. These varied program models are implemented as operators like any other in a relatively few lines of code over the open and extensible XVM API.

```
<edo:model xmlns:edo="ns:com.exocen.edo"
           xmlns:core="ns:com.exocen.core"
       xmlns:mach.out="ns:com.ex..mach.ps...
           xmlns:node="ns:com.exocen.Node">

 <core:function id="fsm">
  <fsm>
   <state>0
    <node:child-text-set>1</...>
    <core:echo/>
   </state>
   <state>1
    <node:child-text-set>0</...>
    <core:echo/>
   </state>
  </fsm>
 </core:function>

 <node:push src="rel:fsm.xml"/>

 <core:for index="0" many="10">

  <core:call src="rel:#fsm"/>
 </core:for>

</edo:model>
```

The basic XVM interprets command line programs, while server and client XVMs extend this one with network service and desktop graphics feature sets respectively.

Additional Edo 6 programming language features include include inheritance and access control.

4 Presentation layer

We like to think about the place for XVM applications in terms of a presentation layer in the network application infrastructure. XVM applications have great strengths, and particular weaknesses. We don't want to implement a database in the Edo 6 code, for example, but we certainly want to access all kinds of stores and services.

High development productivity over a Service Oriented Architecture (SOA) or RDBMS backend is a great strength. Designing and implementing XVM

applications for their configuration management is natural and easy. Using Edo 6 application code as a process model is also natural, likewise visualizing programs and meta data in a variety of styles or models, or for varied media including Web, Mobile, AJAX and Services. Integrating any kind of content format parsers and builders, and network protocol drivers is as easy as could be.

The XVM programming and modelling artefacts in the presentation layer have far greater inherent value than traditional programming code. Their visualization, management, communication, transformation and analysis is completely practical. XVM operators should (may) have completely stable syntax and semantics, due to the virtue of their implementation layer to absorb target API changes. So the lifespan of XVM programming artefacts is far longer than that of traditional program code.

For all these advantages, the *pièce de resistance* is the open, regular and uniform world of data code modelling. An XVM data structure is an XML document open to expansion, and accessable from everywhere. The power of this simple fact, exploited by first class network references in Edo 6, makes this system a distinct departure from the pains of traditional application development platforms.

References

[1] S. Abiteboul, O. Benjelloun, and T. Milo. Positive Active XML, 2004.

[2] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.

[3] M. Hennessey and Robin Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, (32):137–162, 1985.

[4] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[5] Quanzhong Li, Michelle Y. Kim, Edward So, and Steve Wood. XVM: XML virtual machine. *ACM SAC*, 2004.

[6] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer Verlag, New York, 1992.

[7] John von Neumann. *Theory of Self Reproducing Automata*. University of Illinois Press, 1966.

[8] Peter Wegner. Interactive software technology. In *Allen B. Tucker, Jr. (Editor-in-Chief), The Computer Science and Engineering Handbook, CRC Press, in cooperation with ACM, 1997*. 1997.

[9] Peter Wegner. Why interaction is more powerful than algorithms. *Communications of the ACM*, 40(5):80–91, 1997.

[10] Peter Wegner. Interactive foundations of computing. *Theoretical Computer Science*, 192(2):315–351, 1998.

[11] Peter Wegner and Dina Goldin. Interaction as a framework for modeling. *Lecture Notes in Computer Science*, 1565:243–257, 1999.